

Artificial Intelligence Neural Network System Consciousness Perception Ability and Agile Five Finger Peak Core Code "2025 v1.1

PATHON,C++,JAVA,①②③④⑤⑥

● 1. • LIF (Leaky Integrate-and-Fire) • 1000Hz (1ms) • <5ms • $10^6/\text{cm}^3$ () 2. • 4K@120fps • $1000/\text{cm}^2$ • 0.01° ① Python/PyTorch python import torch import torch.nn as nn class BioInspiredNN(nn.Module): def __init__(self, input_dim=128, hidden_dim=1024, motor_dim=24): super().__init__() # self.thalamic_gate = nn.LSTM(input_dim, hidden_dim, bidirectional=True) # self.cerebellum = nn.Sequential(nn.Conv1d(hidden_dim*2, 512, kernel_size=3), nn.ReLU(), nn.Dropout(0.2), nn.MaxPool1d(2)) # self.motor_cortex = nn.Linear(512, motor_dim) def forward(self, sensory_input): # (1ms) temporal_features, _ = self.thalamic_gate(sensory_input) # spatial_features = self.cerebellum(temporal_features.permute(0,2,1)) # motor_output = self.motor_cortex(spatial_features.squeeze(2)) return torch.sigmoid(motor_output) # ② C++/Eigen cpp #include <Eigen/Dense> using namespace Eigen; class BioChemicalModel { public: MatrixXd simulateProteinFolding(const VectorXd& geneSequence) { // const int n = geneSequence.size(); MatrixXd protein(n, 3); protein.row(0) = Vector3d::Zero(); for (int i=1; i<n; i++) { Vector3d prev = protein.row(i-1); double angle = geneSequence[i] * M_PI; protein.row(i) = prev + Vector3d(cos(angle), sin(angle), 0); } return protein; } void neuronSignalPropagation(MatrixXd& ionConcentrations) { // double dt = 0.001; // 1ms MatrixXd gradients = ionConcentrations.unaryExpr([&](double x){ return 1/(1+exp(-x)); }); ionConcentrations += dt * (ionConcentrations.cwiseProduct(gradients)); }; ④ (Java) java public class NeuroMuscularController { private static final int JOINTS = 24; private final double[][] proprioceptiveWeights; public NeuroMuscularController(double[][] weights) { this.proprioceptiveWeights = weights; } public double[] computeMotorCommand(double[] sensoryInput) { double[] motorOutput = new

```
double[JOINTS]; // 50µs for (int j=0; j<JOINTS; j++) { for (int s=0;
s<sensoryInput.length; s++) { motorOutput[j] += sensoryInput[s] *
proprioceptiveWeights[s][j]; } // motorOutput[j] =
sigmoid(motorOutput[j]); } return motorOutput; } private double sigmoid(double
x) { return 1 / (1 + Math.exp(-x)); }}
```

sql -- CREATE TABLE NeuroMotorMapping (neuron_id UUID PRIMARY KEY, joint_id INT NOT NULL, activation_threshold DECIMAL(4,3), connection_strength DECIMAL(3,2) CHECK (connection_strength BETWEEN 0 AND 1), neurotransmitter_type VARCHAR(20) CHECK (neurotransmitter_type IN ('GABA', 'Glutamate', 'Acetylcholine')), last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP);-- CREATE TABLE MultimodalPerception (perception_id SERIAL PRIMARY KEY, visual_feature_vector FLOAT[512], tactile_feature_vector FLOAT[128], auditory_feature_vector FLOAT[256], fused_embedding FLOAT[1024] GENERATED ALWAYS AS (-- visual_feature_vector || tactile_feature_vector || auditory_feature_vector) STORED); graph LR[A --> B{ } B --> C[] C --> D[] D --> E[] E --> F[] F --> G[] G --> H[] H --> I[] I --> A]

1. Memristor • (SNN) 2. python class HierarchicalRL: def __init__(self): self.meta_controller = Transformer() # self.meta_controller = LSTM() # def train(self, sensory_stream): # for t in sensory_stream: abstract_goal = self.meta_controller.predict(t) motor_command = self.meta_controller.predict(t, abstract_goal) reward = env.execute(motor_command) self.update(reward) # 3. • ECoG (256/mm²) • 130-350Hz • <10ms 0.01mm • <50ms 0.1° • <20ms 0.05mm • <5ms 0.5° • 1. 2. 3. 4. AI FPGA/

```
python# Python - (PyTorch)import torchimport torch.nn as nnclass CerebellumNetwork(nn.Module): def __init__(self, neuron_count=1024): super().__init__() self.sensory_layer = nn.Linear(256, neuron_count) self.interneuron = nn.LSTM(neuron_count, neuron_count*2) self.motor_output = nn.Linear(neuron_count*2, 64) def forward(self, x): x = torch.relu(self.sensory_layer(x)) x, _ = self.interneuron(x) return self.motor_output(x) # 1024/1-5ms<0.5W`cpp// C++ - (ROS 2)#include <rclcpp/rclcpp.hpp>#include <sensor_msgs/msg/joint_state.hpp>class ArticulatedHand : public rclcpp::Node { public: ArticulatedHand() : Node("hand_controller") { joint_pub_ = this->create_publisher<sensor_msgs::msg::JointState>("joint_states", 10); timer_ = this->create_wall_timer( std::chrono::milliseconds(5), { this->timer_callback(); }); } private: void timer_callback() { auto msg = sensor_msgs::msg::JointState(); // 7-DOF for(int i=0; i<5; i++) { joints_[i] = calculate_inverse_kinematics(i); } msg.position = joints_; joint_pub_->publish(msg); } // 200Hz±0.01° 0.1N};`java// Java - public class MultiModalFusion { private ConcurrentMap<String, SensorData> sensorCache; public void
```

```
processBioSignal(BioSignal signal) { // OSGi 10GB/s<10ms} `` (PostgreSQL) `` sql-- CREATE TABLE
neuron_connections ( pre_synaptic UUID NOT NULL, post_synaptic UUID NOT
NULL, weight FLOAT CHECK (weight BETWEEN 0 AND 1), delay INT CHECK (delay
BETWEEN 1 AND 20), plasticity BOOLEAN, CONSTRAINT pk_connections PRIMARY
KEY (pre_synaptic, post_synaptic));-- CREATE TABLE joint_parameters
( joint_id SERIAL PRIMARY KEY, joint_type VARCHAR(20) CHECK (joint_type IN
('revolute', 'prismatic', 'spherical')), min_angle FLOAT, max_angle FLOAT,
max_velocity FLOAT, torque_coeff FLOAT[3], calibration_data JSONB); ``
|  |  |  |  ||-----|-----| | | | | | |
|-----||  |  |  | 1.2×106  ||  |  | 104  |
|  |  | 0.8W/103  ||  |  | 5-7 DOF ||  |  | ±0.005° ||  |
|  | 2kHz ||  | 10ns/step ||  | 1μs/  ||  |  |
|  | <10μs ||  | 25Gbps | 1. NEURON+Python  BlueBrain 2. ROS 2 Galactic 3. Apache Arrow 4. Transformer-XL+GNN
1. M1 ANN 2. FPGA (<1ms)3. Loihi 2 4. TMR)+
```

● Python - `pythonimport tensorflow as tfimport numpy as npclass BrainNeuralNetwork: def __init__(self): self.sensory_cortex = self.build_sensory_network() self.motor_cortex = self.build_motor_network() self.cerebellum = self.build_cerebellum() def build_sensory_network(self): """ model = tf.keras.Sequential([tf.keras.layers.Conv3D(64, (3,3,3), activation='relu', input_shape=(128,128,64,3)), tf.keras.layers.BatchNormalization(), tf.keras.layers.LSTM(128, return_sequences=True), tf.keras.layers.Attention(), tf.keras.layers.Dense(512, activation='sigmoid')]) return model def build_motor_network(self): """ model = tf.keras.Sequential([tf.keras.layers.Dense(1024, activation='relu'), tf.keras.layers.Dropout(0.3), tf.keras.layers.Dense(512, activation='tanh'), tf.keras.layers.Dense(256, activation='linear')]) return model `` C++ - #include <vector>#include <memory>#include <eigen3/Eigen/Dense>class NeuralProcessor {private: std::vector<Eigen::MatrixXd> weights; std::vector<Eigen::VectorXd> biases;public: NeuralProcessor(int input_size, int hidden_size, int output_size) { // weights.push_back(Eigen::MatrixXd::Random(hidden_size, input_size)); weights.push_back(Eigen::MatrixXd::Random(output_size, hidden_size)); biases.push_back(Eigen::VectorXd::Zero(hidden_size)); biases.push_back(Eigen::VectorXd::Zero(output_size)); } Eigen::VectorXd forward(const Eigen::VectorXd& input) { Eigen::VectorXd hidden = (weights[0] * input + biases[0]).array().tanh(); Eigen::VectorXd output = (weights[1] * hidden + biases[1]).array().sigmoid(); return output; } };class CerebellumModule`

```

{public: void fineMotorControl(const std::vector<double>& sensor_data,
std::vector<double>& motor_commands) { // ① for(size_t i = 0; i <
sensor_data.size(); ++i) { motor_commands[i] = sensor_data[i] * 0.8 +
calculateFeedback(sensor_data, i) * 0.2; } }};``② ③ Java ④ - ⑤
``javapublic class BioChemicalProcessor { private Map<String, Double>
enzymeLevels; private Map<String, Double> proteinConcentrations; private
NeuralNetwork biochemicalNetwork; public BioChemicalProcessor()
{ this.enzymeLevels = new HashMap<>(); this.proteinConcentrations = new
HashMap<>(); initializeBiochemicalNetwork(); } private void
initializeBiochemicalNetwork() { // ⑥ int[] layers = {128, 64, 32, 16};
this.biochemicalNetwork = new NeuralNetwork(layers); } public void
processNeurotransmitters(double[] neurotransmitters) { // ⑦ double[]
processedSignals = biochemicalNetwork.forward(neurotransmitters);
updateEnzymeActivity(processedSignals);
regulateProteinSynthesis(processedSignals); } public class
GeneExpressionModule { public void expressProteins(String geneSequence,
double expressionLevel) { // ⑧ System.out.println("Expressing protein
from gene: " + geneSequence); } }};``⑨ ⑩ Python ⑪ - ⑫
``pythonclass CardiovascularSystem: def __init__(self): self.heart_rate = 72 #
BPM self.blood_pressure = (120, 80) # systolic/diastolic self.oxygen_level = 98 #
percentage def monitor_vital_signs(self): """⑬""" return { 'heart_rate':
self.heart_rate, 'blood_pressure': self.blood_pressure, 'oxygen_saturation':
self.oxygen_level, 'cardiac_output': self.calculate_cardiac_output() } def
regulate_circulation(self, activity_level): """⑭""" if activity_level > 0.7:
self.heart_rate = min(180, self.heart_rate + 20) elif activity_level < 0.3:
self.heart_rate = max(60, self.heart_rate - 10)``⑮ ⑯ C++ ⑰ - ⑱
⑲ ``cppclass HandKinematics {private: struct Joint { double position; double
velocity; double torque; double stiffness; }; std::vector<Joint> finger_joints;
const int JOINTS_PER_FINGER = 3; const int FINGER_COUNT = 5;public:
HandKinematics() { initializeJoints(); } void initializeJoints()
{ finger_joints.resize(FINGER_COUNT * JOINTS_PER_FINGER); for(auto& joint :
finger_joints) { joint.position = 0.0; joint.velocity = 0.0; joint.torque = 0.0;
joint.stiffness = 1.0; } } std::vector<double> calculateInverseKinematics( const
Eigen::Vector3d& target_position, int finger_index) { std::vector<double>
joint_angles(JOINTS_PER_FINGER, 0.0); // ⑲ // ⑲ - ⑲ for(int i
= 0; i < JOINTS_PER_FINGER; ++i) { joint_angles[i] = target_position[i] * (i + 1) *
0.1; } return joint_angles; }};class TactileSensorNetwork {public: struct
TactileData { double pressure; double vibration; double temperature; double
texture; }; std::vector<TactileData> sensor_readings; void
processTactileFeedback() { for(auto& sensor : sensor_readings) { // ⑳
if(sensor.pressure > 50.0) { triggerGripAdjustment(sensor); } } }};``㉑ ㉒
㉒ Java ㉒ - ㉒ ``javapublic class MultimodalFusionSystem { private
BrainNeuralNetwork brain; private HandKinematics hand; private
CardiovascularSystem heart; private BioChemicalProcessor biochemistry; public
MultimodalFusionSystem() { this.brain = new BrainNeuralNetwork(); this.hand =
new HandKinematics(); this.heart = new CardiovascularSystem();
this.biochemistry = new BioChemicalProcessor(); } public class

```

```

SensorFusionModule { public double[] fuseSensoryInputs(double[] visual,
double[] tactile, double[] proprioceptive) { // 融合感觉 double[] fused = new
double[visual.length]; for(int i = 0; i < visual.length; i++) { fused[i] = (visual[i] *
0.4 + tactile[i] * 0.4 + proprioceptive[i] * 0.2); } return fused; } } public class
AdaptiveLearningModule { public void reinforceSuccessfulMovements(double[]
motor_commands, double[] sensory_feedback) { // 强化学习 double error =
calculateError(motor_commands, sensory_feedback); if(error < 0.1)
{ reinforcePathway(motor_commands); } } }
Python 代码 - 手术机器人系统
pythonclass SurgicalRobotSystem: def __init__(self): self.brain_system =
BrainNeuralNetwork() self.hand_control = HandKinematics() self.safety_monitor
= SafetySystem() def perform_delicate_surgery(self, target_anatomy,
procedure_plan): """规划手术路径""" # 规划手术路径 surgical_path =
self.plan_surgical_path(target_anatomy) # 手术路径 while not
procedure_complete: sensory_data = self.acquire_sensory_data()
motor_commands = self.brain_system.process(sensory_data) # 处理 if
self.safety_monitor.check_safety(motor_commands):
self.hand_control.execute_movement(motor_commands) else:
self.trigger_safety_stop()
yaml# 神经网络: layers:
[1024, 512, 256, 128] activation: "swish" learning_rate: 0.001 dropout_rate:
0.3MotorControl: update_frequency: 1000Hz control_latency: <2ms precision:
0.01mm max_torque: 50NmSensorySystem: tactile_resolution: 0.1mm
pressure_range: 0-1000kPa temperature_range: 0-100°C sampling_rate:
2000HzBiochemical: processing_delay: <10ms signal_amplification: 1000x
noise_rejection: 60dB
1. 手术机器人系统 2. 手术机器人系统-手术机器人系统
3. 手术机器人系统 4. 手术机器人系统 5. 手术机器人系统
手术机器人系统

```

●手术机器人系统 ①

```

Python 代码 - 手术机器人系统
pythonimport tensorflow as tfimport numpy
as npclass BrainNeuralNetwork: def __init__(self): self.sensory_cortex =
self.build_sensory_network() self.motor_cortex = self.build_motor_network()
self.cerebellum = self.build_cerebellum() def build_sensory_network(self): """构建
神经网络""" model = tf.keras.Sequential([ tf.keras.layers.Conv3D(64, (3,3,3),
activation='relu', input_shape=(128,128,64,3)),
tf.keras.layers.BatchNormalization(), tf.keras.layers.LSTM(128,
return_sequences=True), tf.keras.layers.Attention(), tf.keras.layers.Dense(512,
activation='sigmoid') ]) return model def build_motor_network(self): """构建
神经网络""" model = tf.keras.Sequential([ tf.keras.layers.Dense(1024, activation='relu'),
tf.keras.layers.Dropout(0.3), tf.keras.layers.Dense(512, activation='tanh'),
tf.keras.layers.Dense(256, activation='linear') ]) return model
C++ 代码 - 手术机器人系统
cpp#include <vector>#include <memory>#include
<Eigen3/Eigen/Dense>class NeuralProcessor {private:
std::vector<Eigen::MatrixXf> weights; std::vector<Eigen::VectorXf>
biases;public: NeuralProcessor(int input_size, int hidden_size, int output_size) { //
初始化 weights.push_back(Eigen::MatrixXf::Random(hidden_size,
input_size)); weights.push_back(Eigen::MatrixXf::Random(output_size,
hidden_size)); biases.push_back(Eigen::VectorXf::Zero(hidden_size));

```

```

biases.push_back(Eigen::VectorXd::Zero(output_size)); } Eigen::VectorXd
forward(const Eigen::VectorXd& input) { Eigen::VectorXd hidden = (weights[0] *
input + biases[0]).array().tanh(); Eigen::VectorXd output = (weights[1] * hidden
+ biases[1]).array().sigmoid(); return output; } };class CerebellumModule
{public: void fineMotorControl(const std::vector<double>& sensor_data,
std::vector<double>& motor_commands) { // ① for(size_t i = 0; i <
sensor_data.size(); ++i) { motor_commands[i] = sensor_data[i] * 0.8 +
calculateFeedback(sensor_data, i) * 0.2; } } };``② Python ③ Python ④ C++ ⑤ Java ⑥ Java
``javapublic class BioChemicalProcessor { private Map<String, Double>
enzymeLevels; private Map<String, Double> proteinConcentrations; private
NeuralNetwork biochemicalNetwork; public BioChemicalProcessor()
{ this.enzymeLevels = new HashMap<>(); this.proteinConcentrations = new
HashMap<>(); initializeBiochemicalNetwork(); } private void
initializeBiochemicalNetwork() { // int[] layers = {128, 64, 32, 16};
this.biochemicalNetwork = new NeuralNetwork(layers); } public void
processNeurotransmitters(double[] neurotransmitters) { // double[]
processedSignals = biochemicalNetwork.forward(neurotransmitters);
updateEnzymeActivity(processedSignals);
regulateProteinSynthesis(processedSignals); } public class
GeneExpressionModule { public void expressProteins(String geneSequence,
double expressionLevel) { // System.out.println("Expressing protein
from gene: " + geneSequence); } } } ``③ Python ④ C++ ⑤ Java ⑥ Java
``pythonclass CardiovascularSystem: def __init__(self): self.heart_rate = 72 #
BPM self.blood_pressure = (120, 80) # systolic/diastolic self.oxygen_level = 98 #
percentage def monitor_vital_signs(self): """ return { 'heart_rate':
self.heart_rate, 'blood_pressure': self.blood_pressure, 'oxygen_saturation':
self.oxygen_level, 'cardiac_output': self.calculate_cardiac_output() } def
regulate_circulation(self, activity_level): """ if activity_level > 0.7:
self.heart_rate = min(180, self.heart_rate + 20) elif activity_level < 0.3:
self.heart_rate = max(60, self.heart_rate - 10)``④ C++ ⑤ Java ⑥ Java
``cppclass HandKinematics {private: struct Joint { double position; double
velocity; double torque; double stiffness; }; std::vector<Joint> finger_joints;
const int JOINTS_PER_FINGER = 3; const int FINGER_COUNT = 5;public:
HandKinematics() { initializeJoints(); } void initializeJoints()
{ finger_joints.resize(FINGER_COUNT * JOINTS_PER_FINGER); for(auto& joint :
finger_joints) { joint.position = 0.0; joint.velocity = 0.0; joint.torque = 0.0;
joint.stiffness = 1.0; } } std::vector<double> calculateInverseKinematics( const
Eigen::Vector3d& target_position, int finger_index) { std::vector<double>
joint_angles(JOINTS_PER_FINGER, 0.0); // for(int i = 0; i < JOINTS_PER_FINGER; ++i) { joint_angles[i] = target_position[i] * (i + 1) *
0.1; } return joint_angles; } };class TactileSensorNetwork {public: struct
TactileData { double pressure; double vibration; double temperature; double
texture; }; std::vector<TactileData> sensor_readings; void
processTactileFeedback() { for(auto& sensor : sensor_readings) { if(sensor.pressure > 50.0) { triggerGripAdjustment(sensor); } } } };``⑤ Java ⑥ Java
``javapublic class MultimodalFusionSystem { private
BrainNeuralNetwork brain; private HandKinematics hand; private

```

```

CardiovascularSystem heart; private BioChemicalProcessor biochemistry; public
MultimodalFusionSystem() { this.brain = new BrainNeuralNetwork(); this.hand =
new HandKinematics(); this.heart = new CardiovascularSystem();
this.biochemistry = new BioChemicalProcessor(); } public class
SensorFusionModule { public double[] fuseSensoryInputs(double[] visual,
double[] tactile, double[] proprioceptive) { // 融合感觉输入 double[] fused = new
double[visual.length]; for(int i = 0; i < visual.length; i++) { fused[i] = (visual[i] *
0.4 + tactile[i] * 0.4 + proprioceptive[i] * 0.2); } return fused; } } public class
AdaptiveLearningModule { public void reinforceSuccessfulMovements(double[]
motor_commands, double[] sensory_feedback) { // 强化成功运动 double error =
calculateError(motor_commands, sensory_feedback); if(error < 0.1)
{ reinforcePathway(motor_commands); } } } ```` ⑥ Python 代码 - 手术机器人
系统 ````pythonclass SurgicalRobotSystem: def __init__(self): self.brain_system =
BrainNeuralNetwork() self.hand_control = HandKinematics() self.safety_monitor
= SafetySystem() def perform_delicate_surgery(self, target_anatomy,
procedure_plan): """手术计划""" # 规划手术路径 surgical_path =
self.plan_surgical_path(target_anatomy) # 开始手术 while not
procedure_complete: sensory_data = self.acquire_sensory_data()
motor_commands = self.brain_system.process(sensory_data) # 执行 if
self.safety_monitor.check_safety(motor_commands):
self.hand_control.execute_movement(motor_commands) else:
self.trigger_safety_stop() ```` 配置文件 ````yaml# 神经网络: layers:
[1024, 512, 256, 128] activation: "swish" learning_rate: 0.001 dropout_rate:
0.3MotorControl: update_frequency: 1000Hz control_latency: <2ms precision:
0.01mm max_torque: 50NmSensorySystem: tactile_resolution: 0.1mm
pressure_range: 0-1000kPa temperature_range: 0-100°C sampling_rate:
2000HzBiochemical: processing_delay: <10ms signal_amplification: 1000x
noise_rejection: 60dB ```` 1. 系统初始化 2. 获取手术目标
3. 规划手术路径 4. 开始手术 5. 实时监控手术过程

```

●数据库设计MySQL数据库设计①②③④⑤⑥数据库设计-- 1. 创建数据库①②CREATE TABLE brain_neural (id INT PRIMARY KEY AUTO_INCREMENT, neural_type VARCHAR(50) NOT NULL COMMENT '神经网络类型', synapse_count INT NOT NULL COMMENT '突触数量', bio_chemical_param JSON NOT NULL COMMENT '生化参数/物理参数', physical_conductivity FLOAT NOT NULL COMMENT '物理导电性(0-1)', feedback_delay FLOAT NOT NULL COMMENT '反馈延迟(ms)', create_time DATETIME DEFAULT CURRENT_TIMESTAMP);-- 2. 创建数据库④CREATE TABLE hand_joint (id INT PRIMARY KEY AUTO_INCREMENT, joint_name VARCHAR(30) NOT NULL COMMENT '关节名称/类型', neural_connection_count INT NOT NULL COMMENT '神经连接数', motion_precision FLOAT NOT NULL COMMENT '运动精度(mm)', torque_range FLOAT NOT NULL COMMENT '扭矩范围(N·m)', bio_param JSON NOT NULL COMMENT '生化参数/物理参数', status TINYINT DEFAULT 1 COMMENT '状态0-正常/1-异常', update_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);-- 3. 创建数据库①⑥CREATE TABLE multimodal_data (id INT PRIMARY KEY AUTO_INCREMENT, scene_type VARCHAR(50) NOT NULL COMMENT '场景类型/感知类型', perception_type VARCHAR(30) NOT NULL

```

COMMENT 'YYYYMMDD/HH/MM', data_content BLOB NOT NULL COMMENT 'YYYYMMDD',
neural_process_result JSON NOT NULL COMMENT 'YYYYMMDD', feedback_control_cmd
VARCHAR(100) NOT NULL COMMENT 'YYYYMMDD', collect_time DATETIME DEFAULT
CURRENT_TIMESTAMP);-- 4. 创建系统优化表CREATE TABLE system_optimization
( id INT PRIMARY KEY AUTO_INCREMENT, optimization_target VARCHAR(50) NOT
NULL COMMENT 'YYYYMMDD/HH/MM', brain_hand_weight FLOAT NOT NULL COMMENT
'YYYYMMDD(0-1)', neural_adjust_coeff FLOAT NOT NULL COMMENT 'YYYYMMDD',
joint_response_threshold FLOAT NOT NULL COMMENT 'YYYYMMDD', effective_flag
TINYINT DEFAULT 1 COMMENT 'YYYYMMDD0-1/1-0'); 5. Python/C++/Java1.
Python 使用TensorFlow和MySQLimport tensorflow as tfimport pymysqlimport
jsonfrom datetime import datetime# 创建数据库和表①②class
BrainNeuralModule: def __init__(self, neural_type=""): self.neural_type =
neural_type self.model = self.build_neural_network() # 创建神经网络 self.db_conn =
self.init_db() # 创建数据库连接 # 3. 创建神经网络def build_neural_network(self):
model = tf.keras.Sequential([ tf.keras.layers.Dense(256, activation='relu',
input_shape=(128,)), tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(64, activation='sigmoid') # 创建神经网络 ])
model.compile(optimizer='adam', loss='mse') return model # 返回模型def
init_db(self): conn = pymysql.connect( host="localhost",
user="root", password="123456", database="ai_brain_hand", charset="utf8" )
return conn # 返回数据库连接def process_signal(self, perception_data):
# 1. 创建游标 cursor = self.db_conn.cursor() cursor.execute("SELECT
bio_chemical_param, physical_conductivity FROM brain_neural WHERE
neural_type=%s", self.neural_type) bio_param, conductivity = cursor.fetchone()
bio_param = json.loads(bio_param) # 2. 处理感知数据 processed_data =
self.model.predict(perception_data.reshape(1, -1))[0] # 3. 计算反馈信号
feedback_signal = processed_data * conductivity return feedback_signal# 4. 创建
HandJointModuleclass HandJointModule: def __init__(self): self.db_conn =
pymysql.connect( host="localhost", user="root", password="123456",
database="ai_brain_hand", charset="utf8" ) # 创建数据库连接def
control_joint(self, feedback_signal, joint_name=""): cursor =
self.db_conn.cursor() # 创建游标 cursor.execute("SELECT motion_precision,
torque_range FROM hand_joint WHERE joint_name=%s", joint_name) precision,
torque = cursor.fetchone() # 计算关节角度 joint_angle =
(feedback_signal.mean() * 90) / torque # 限制角度在0-90度之间
cursor.execute( "INSERT INTO multimodal_data (scene_type, perception_type,
feedback_control_cmd) VALUES (%s, %s, %s)", ("BrainNeural", joint_name, f"{
joint_name},{joint_angle:.2f},{precision:.3f}") )
self.db_conn.commit() return {"joint_name": joint_name, "angle": joint_angle,
"precision": precision}# 5. 6. 主程序if __name__ == "__main__":
brain_module = BrainNeuralModule() hand_module = HandJointModule() # 创建模块
perception_data = tf.random.normal((128,)) # 生成感知数据
feedback_signal = brain_module.process_signal(perception_data) # 计算反馈信号
control_result = hand_module.control_joint(feedback_signal) print("反馈信号",
control_result) 2. C++ 使用MySQL和Eigen#include <iostream>#include
<mysql_driver.h>#include <mysql_connection.h>#include
<json/json.h>#include <Eigen/Dense> // 使用命名空间

```



```

std;using namespace Eigen;// ①const string DB_HOST = "localhost";const
string DB_USER = "root";const string DB_PWD = "123456";const string DB_NAME
= "ai_brain_hand";// ②class BrainNeuralModule {private:
sql::mysql::MySQL_Driver* driver; sql::Connection* conn; MatrixXd weights1,
weights2, weights3; // ③public: // ④+⑤BrainNeuralModule()
{ // ⑥driver = sql::mysql::get_mysql_driver_instance(); conn = driver-
>connect(DB_HOST, DB_USER, DB_PWD); conn->setSchema(DB_NAME); // ⑦
⑧weights1 = MatrixXd::Random(256, 128) * 0.1; weights2 =
MatrixXd::Random(128, 256) * 0.1; weights3 = MatrixXd::Random(64, 128) * 0.1;
} // ⑨VectorXd processSignal(const VectorXd& perceptionData)
{ // 1. ⑩sql::Statement* stmt = conn->createStatement(); sql::ResultSet*
res = stmt->executeQuery("SELECT bio_chemical_param, physical_conductivity
FROM brain_neural WHERE neural_type='0'"); res->next(); string bioParamStr =
res->getString("bio_chemical_param"); float conductivity = res-
>getDouble("physical_conductivity"); // 2. ⑪VectorXd layer1 = (weights1
* perceptionData).unaryExpr([](double x) { return max(0.0, x); }); // ReLU
VectorXd layer2 = (weights2 * layer1).unaryExpr([](double x) { return max(0.0,
x); }); VectorXd layer3 = (weights3 * layer2).unaryExpr([](double x) { return
1.0 / (1.0 + exp(-x)); }); // Sigmoid // 3. ⑫VectorXd feedbackSignal =
layer3 * conductivity; delete stmt; delete res; return feedbackSignal; }
~BrainNeuralModule() { delete conn; };// ⑬class HandJointModule
{private: sql::mysql::MySQL_Driver* driver; sql::Connection* conn;public:
HandJointModule() { driver = sql::mysql::get_mysql_driver_instance(); conn =
driver->connect(DB_HOST, DB_USER, DB_PWD); conn-
>setSchema(DB_NAME); } // ⑭void controlJoint(const
VectorXd& feedbackSignal, const string& jointName) { // ⑮
sql::PreparedStatement* pstmt = conn->prepareStatement( "SELECT
motion_precision, torque_range FROM hand_joint WHERE joint_name=?" ); pstmt-
>setString(1, jointName); sql::ResultSet* res = pstmt->executeQuery(); res-
>next(); float precision = res->getDouble("motion_precision"); float torque =
res->getDouble("torque_range"); // ⑯float avgSignal =
feedbackSignal.mean(); float jointAngle = (avgSignal * 90.0) / torque; // ⑰
sql::PreparedStatement* logStmt = conn->prepareStatement( "INSERT INTO
multimodal_data (scene_type, perception_type, feedback_control_cmd) VALUES
(?, ?, ?)" ); string cmd = "0=" + jointName + ",0=" + to_string(jointAngle) + ",
0=" + to_string(precision); logStmt->setString(1, "0000"); logStmt->setString(2,
"00+00"); logStmt->setString(3, cmd); logStmt->execute(); cout << "0000000"
<< cmd << endl; delete pstmt; delete res; delete logStmt; }
~HandJointModule() { delete conn; };// ⑱int main()
{ BrainNeuralModule brainModule; HandJointModule handModule; // ⑲128 ⑳
VectorXd perceptionData = VectorXd::Random(128); // ㉑
VectorXd feedbackSignal = brainModule.processSignal(perceptionData); // ㉒
handModule.controlJoint(feedbackSignal, "00"); return 0;} 3. Java ㉓
import com.mysql.cj.jdbc.MysqlDataSource;import
org.deeplearning4j.nn.conf.MultiLayerConfiguration;import
org.deeplearning4j.nn.conf.NeuralNetConfiguration;import
org.deeplearning4j.nn.conf.layers.DenseLayer;import

```

```

org.deeplearning4j.nn.conf.layers.OutputLayer;import
org.deeplearning4j.nn.multilayer.MultiLayerNetwork;import
org.nd4j.linalg.activations.Activation;import
org.nd4j.linalg.api.ndarray.INDArray;import org.nd4j.linalg.factory.Nd4j;import
org.nd4j.linalg.lossfunctions.LossFunctions;import org.json.JSONObject;import
java.sql.Connection;import java.sql.PreparedStatement;import
java.sql.ResultSet;import java.sql.SQLException;// 数据库类 DBUtil { private
static final String URL = "jdbc:mysql://localhost:3306/ai_brain_hand?
useSSL=false"; private static final String USER = "root"; private static final String
PWD = "123456"; public static Connection getConn() throws SQLException
{ MysqlDataSource dataSource = new MysqlDataSource();
dataSource.setURL(URL); dataSource.setUser(USER);
dataSource.setPassword(PWD); return dataSource.getConnection(); } }// 数据库类
// ① 类 BrainNeuralModule { private MultiLayerNetwork neuralNetwork;
private Connection conn; public BrainNeuralModule() throws SQLException { // 初始化
MultiLayerConfiguration config = new
NeuralNetConfiguration.Builder() .updater(new
org.nd4j.linalg.learning.config.Sgd(0.01)) .list() .layer(new
DenseLayer.Builder().nIn(128).nOut(256).activation(Activation.RELU).build()) .layer(new
DenseLayer.Builder().nIn(256).nOut(128).activation(Activation.RELU).build()) .layer(new
OutputLayer.Builder(LossFunctions.LossFunction.MSE) .nIn(128).nOut(64).activation(Activation.SIGMOID).build()) .build(); neuralNetwork = new
MultiLayerNetwork(config); neuralNetwork.init(); // 数据库 conn =
DBUtil.getConn(); } // 处理信号 public INDArray processSignal(INDArray
perceptionData) throws SQLException { // 数据库 PreparedStatement pstmt =
conn.prepareStatement( "SELECT bio_chemical_param, physical_conductivity
FROM brain_neural WHERE neural_type=?" ); pstmt.setString(1, ""); ResultSet
rs = pstmt.executeQuery(); rs.next(); String bioParamStr =
rs.getString("bio_chemical_param"); float conductivity =
rs.getFloat("physical_conductivity"); // 数据库 INDArray processedData =
neuralNetwork.output(perceptionData); // 数据库 INDArray feedbackSignal =
processedData.mul(conductivity); rs.close(); pstmt.close(); return
feedbackSignal; } }// 数据库 ④ 类 HandJointModule { private Connection
conn; public HandJointModule() throws SQLException { conn =
DBUtil.getConn(); } // 数据库 public void controlJoint(INDArray
feedbackSignal, String jointName, String sceneType) throws SQLException { // 数据库
PreparedStatement pstmt = conn.prepareStatement( "SELECT
motion_precision, torque_range FROM hand_joint WHERE joint_name=?" );
pstmt.setString(1, jointName); ResultSet rs = pstmt.executeQuery(); rs.next();
float precision = rs.getFloat("motion_precision"); float torque =
rs.getFloat("torque_range"); // 数据库 float avgSignal =
feedbackSignal.meanNumber().floatValue(); float jointAngle = (avgSignal * 90.0f)
/ torque; // 数据库 PreparedStatement logStmt =
conn.prepareStatement( "INSERT INTO multimodal_data (scene_type,
perception_type, feedback_control_cmd) VALUES (?, ?, ?)" ); String cmd =

```



```

edge_index).relu() return self.conv2(x, edge_index)# ① Python + SciPy +
torch.tensor([S.i, S.j], dtype=torch.long)x = torch.randn(num_neurons, 10) # ②
Python + SciPy + SBML
python# biochem_model.pyfrom scipy.integrate import solve_ivpimport
numpy as np# Michaelis-Menten
def enzyme_kinetics(t, y): S, E, ES, P = y kf
= 1e6 # 1/(M*s) kr = 1e-3 # 1/s kcat = 1e2 # 1/s dS = -kf * S * E + kr * ES dE =
-kf * S * E + kr * ES + kcat * ES dES = kf * S * E - kr * ES - kcat * ES dP = kcat *
ES return [dS, dE, dES, dP]# ③ ECG + HRV
C++ FFT + Cpp
// heart_sync.cpp#include <fftw3.h>#include
<vector>#include <cmath>void computeHRV(const std::vector<double>&
rr_intervals) { int N = rr_intervals.size(); fftw_complex *out = (fftw_complex*)
fftw_malloc(sizeof(fftw_complex) * N); fftw_plan p = fftw_plan_dft_r2c_1d(N,
rr_intervals.data(), out, FFTW_ESTIMATE); fftw_execute(p); double lf = 0, hf = 0;
for (int i = 0; i < N/2; ++i) { double freq = i / (rr_intervals.back() -
rr_intervals.front()); double power = std::abs(out[i][0]) * std::abs(out[i][0]); if
(freq >= 0.04 && freq < 0.15) lf += power; if (freq >= 0.15 && freq < 0.4) hf
+= power; } std::cout << "LF/HF ratio: " << lf / hf << std::endl;
fftw_destroy_plan(p); fftw_free(out);} ④ RL + MPC
Python
PyBullet + Stable-Baselines3
python# hand_control_rl.pyimport pybullet as
pimport pybullet_datafrom stable_baselines3 import PPOfrom gym import Env,
spacesimport numpy as npclass HandEnv(Env): def __init__(self):
super().__init__() self.client = p.connect(p.DIRECT)
p.setAdditionalSearchPath(pybullet_data.getDataPath()) self.hand =
p.loadURDF("shadow_hand.urdf") self.action_space = spaces.Box(low=-1,
high=1, shape=(22,), dtype=np.float32) self.observation_space =
spaces.Box(low=-np.inf, high=np.inf, shape=(44,), dtype=np.float32) def
step(self, action): p.setJointMotorControlArray(self.hand, range(22),
p.POSITION_CONTROL, targetPositions=action) p.stepSimulation() obs =
self._get_obs() reward = -np.linalg.norm(action - self.target) done = False return
obs, reward, done, {} def reset(self): p.resetSimulation() self.hand =
p.loadURDF("shadow_hand.urdf") return self._get_obs()env = HandEnv()model =
PPO("MlpPolicy", env, verbose=1)model.learn(total_timesteps=100000) ⑤
AI
Java
Drools + Prolog
java//
AICompensator.javaimport org.jpl7.*;public class AICompensator { public static
void main(String[] args) { Query q = new Query("consult('rules.pl')");
q.hasSolution(); Query q2 = new Query("safe_action(X, context(hand,
unstable))"); while (q2.hasMoreSolutions()) { System.out.println("Recommended
action: " +
q2.nextSolution().get("X")); } } rules.pl prologsafe_action(grasp_firmly,
context(hand, unstable)).safe_action(slow_down, context(vision, low_light)). ⑥
ROS2 + C++
ROS2
C++
// bhcs_agent.cpp#include
"rclcpp/rclcpp.hpp"#include "sensor_msgs/msg/joint_state.hpp"#include
"std_msgs/msg/float32_multi_array.hpp"class BHCSAgent : public rclcpp::Node
{public: BHCSAgent() : Node("bhcs_agent") { pub_ = this-
>create_publisher<std_msgs::msg::Float32MultiArray>("/hand/command", 10);

```

```

sub_ = this->create_subscription<sensor_msgs::msg::JointState>( "/hand/state",
10, std::bind(&BHCSAgent::callback, this, std::placeholders::_1)); }private: void
callback(const sensor_msgs::msg::JointState::SharedPtr msg)
{ std_msgs::msg::Float32MultiArray cmd; cmd.data = compute_control(msg-
>position); pub_->publish(cmd); } std::vector<float> compute_control(const
std::vector<double>& pos) { std::vector<float> cmd(22, 0.0); for (size_t i = 0; i
< 22; ++i) { cmd[i] = -0.1 * pos[i]; # PD [] } return cmd; }
rclcpp::Publisher<std_msgs::msg::Float32MultiArray>::SharedPtr pub_;
rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr sub_;};int
main(int argc, char** argv) { rclcpp::init(argc, argv);
rclcpp::spin(std::make_shared<BHCSAgent>()); rclcpp::shutdown();}```---
PostgreSQL```sql-- CREATE TABLE neurons ( id
SERIAL PRIMARY KEY, type VARCHAR(20), -- excitatory/inhibitory location
VARCHAR(50), gene_expression JSONB, protein_levels JSONB);-- CREATE
TABLE synapses ( id SERIAL PRIMARY KEY, pre_neuron INT REFERENCES
neurons(id), post_neuron INT REFERENCES neurons(id), weight REAL, delay_ms
REAL, neurotransmitter VARCHAR(20));-- CREATE TABLE enzyme_kinetics (
id SERIAL PRIMARY KEY, enzyme_name VARCHAR(50), substrate_conc REAL,
product_conc REAL, kcat REAL, km REAL, timestamp TIMESTAMPTZ DEFAULT
NOW());```---
① <1ms ② <2% ③ <5ms ④ 0.1N ⑤ <3% ⑥ <10ms --- 1.
Unity/Unreal + ROS2 + PyBullet2. Intel LoihiIBM TrueNorth3.
COBRA + NEURON + SBML4. “” + fMRI5. Nengo + Prolog + GraphQL--- “”> “”
--- -- Docker
Kubernetes- Loihi 2- - -

```

● 1.
• LIF (Leaky Integrate-and-Fire) • 1000Hz (1ms)
• <5ms • 10⁶/cm³ (2. • 4K@120fps • 1000 /cm² • 0.01°④ (Python/PyTorch)
python import torchimport torch.nn as nnclass
BioInspiredNN(nn.Module): def __init__(self, input_dim=128, hidden_dim=1024,
motor_dim=24): super().__init__() # self.thalamic_gate =
nn.LSTM(input_dim, hidden_dim, bidirectional=True) # self.cerebellum
= nn.Sequential(nn.Conv1d(hidden_dim*2, 512, kernel_size=3), nn.ReLU(),
nn.Dropout(0.2), nn.MaxPool1d(2)) # self.motor_cortex = nn.Linear(512,
motor_dim) def forward(self, sensory_input): # (1ms)
temporal_features, _ = self.thalamic_gate(sensory_input) #
spatial_features = self.cerebellum(temporal_features.permute(0,2,1)) #
motor_output = self.motor_cortex(spatial_features.squeeze(2)) return
torch.sigmoid(motor_output) # ② (C++/Eigen)cpp #include
<Eigen/Dense>using namespace Eigen;class BioChemicalModel {public:
MatrixXd simulateProteinFolding(const VectorXd& geneSequence) { //
const int n = geneSequence.size(); MatrixXd protein(n, 3); protein.row(0) =
Vector3d::Zero(); // for (int i=1; i<n; i++) { Vector3d prev =

```

protein.row(i-1); double angle = geneSequence[i] * M_PI; protein.row(i) = prev +
Vector3d(cos(angle), sin(angle), 0); } return protein; } void
neuronSignalPropagation(MatrixXd& ionConcentrations) { // 离子浓度传播 double
dt = 0.001; // 1ms 时间步 MatrixXd gradients = ionConcentrations.unaryExpr([]
(double x){ return 1/(1+exp(-x)); // 梯度计算 }); ionConcentrations += dt *
(ionConcentrations.cwiseProduct(gradients)); } };④ 神经网络 (Java) java public
class NeuroMuscularController { private static final int JOINTS = 24; private final
double[][] proprioceptiveWeights; public NeuroMuscularController(double[][]
weights) { this.proprioceptiveWeights = weights; // 初始化 } public double[]
computeMotorCommand(double[] sensoryInput) { double[] motorOutput = new
double[JOINTS]; // 初始化 (50μs 时间步) for (int j=0; j<JOINTS; j++) { for (int s=0;
s<sensoryInput.length; s++) { motorOutput[j] += sensoryInput[s] *
proprioceptiveWeights[s][j]; } // 计算 motorOutput[j] =
sigmoid(motorOutput[j]); } return motorOutput; } private double sigmoid(double
x) { return 1 / (1 + Math.exp(-x)); } }
数据库创建 sql -- 创建数据库 CREATE
TABLE NeuroMotorMapping ( neuron_id UUID PRIMARY KEY, joint_id INT NOT
NULL, activation_threshold DECIMAL(4,3), connection_strength DECIMAL(3,2)
CHECK (connection_strength BETWEEN 0 AND 1), neurotransmitter_type
VARCHAR(20) CHECK (neurotransmitter_type IN ('GABA', 'Glutamate',
'Acetylcholine')), last_modified TIMESTAMP DEFAULT CURRENT_TIMESTAMP);-- 创建
表 CREATE TABLE MultimodalPerception ( perception_id SERIAL PRIMARY
KEY, visual_feature_vector FLOAT[512], tactile_feature_vector FLOAT[128],
auditory_feature_vector FLOAT[256], fused_embedding FLOAT[1024]
GENERATED ALWAYS AS ( -- 计算 visual_feature_vector ||
tactile_feature_vector || auditory_feature_vector ) STORED);
图论 graph LRA[A --> B{ }B --> C[ ]B --> D[ ]C --> E[ ]D -->
F[ ]E --> G[ ]F --> H[ ]G --> HH --> I[ ]I --> A 1.
• 忆阻器 • SNN 2. python
class HierarchicalRL: def __init__(self): self.meta_controller = Transformer() #
self.meta_controller = LSTM() # def train(self, sensory_stream): #
for t in sensory_stream: abstract_goal = self.meta_controller.predict(t)
motor_command = self.meta_controller.predict(t, abstract_goal) reward =
env.execute(motor_command) self.update(reward) #
3. ECoG (256/mm²) • 130-350Hz 4. 1. 2. 3. 4. AI FPGA/

```

● MySQL 数据库 ①②③④⑤⑥ sql-- 1. 创建数据库 ①② CREATE TABLE
brain_neural_system (id BIGINT PRIMARY KEY AUTO_INCREMENT COMMENT 'ID',
neural_region VARCHAR(30) NOT NULL COMMENT '神经网络/层/区',
synapse_count INT NOT NULL COMMENT '突触数量', bio_chem_params JSON
NOT NULL COMMENT '{key:value}/生物/化学', physical_conductivity FLOAT
NOT NULL COMMENT '物理导电性(0-1)', bio_physical_domain VARCHAR(50)
COMMENT '生物物理域', feedback_delay FLOAT NOT NULL COMMENT '反馈延迟(ms)',
update_time DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE

```

CURRENT_TIMESTAMP);-- 2. ③CREATE TABLE cardiac_system ( id
BIGINT PRIMARY KEY AUTO_INCREMENT COMMENT 'ID', heart_rate INT NOT
NULL COMMENT '心跳(次/分)', blood_oxygen FLOAT NOT NULL COMMENT '血氧饱和度(%)',
blood_pressure JSON NOT NULL COMMENT '血压{收缩压:舒张压}', energy_supply
FLOAT NOT NULL COMMENT '能量供给(0-1)', status TINYINT DEFAULT 1 COMMENT
'系统0-正常/1-异常', collect_time DATETIME DEFAULT CURRENT_TIMESTAMP);-- 3. ④
CREATE TABLE limb_joint_system ( id BIGINT PRIMARY KEY
AUTO_INCREMENT COMMENT 'ID', body_part VARCHAR(30) NOT NULL
COMMENT '身体部位/关节名称', joint_type VARCHAR(20) NOT NULL COMMENT '关节
类型/名称', neural_connection_count INT NOT NULL COMMENT '神经连接数量',
motion_precision FLOAT NOT NULL COMMENT '运动精度(mm)', torque_range FLOAT
NOT NULL COMMENT '扭矩范围(N·m)', bio_params JSON NOT NULL COMMENT '生物参数
/生化参数', response_time FLOAT NOT NULL COMMENT '响应时间(ms)', create_time
DATETIME DEFAULT CURRENT_TIMESTAMP);-- 4. ④⑥CREATE
TABLE multimodal_fusion ( id BIGINT PRIMARY KEY AUTO_INCREMENT COMMENT
'ID', application_scene VARCHAR(50) NOT NULL COMMENT '应用场景/环境',
perception_types VARCHAR(100) NOT NULL COMMENT '感知类型+视觉+听觉+触觉',
raw_data BLOB NOT NULL COMMENT '原始数据', neural_process_result JSON NOT
NULL COMMENT '神经处理结果', control_command VARCHAR(200) NOT NULL COMMENT
'控制命令', system_efficiency FLOAT COMMENT '系统效率(0-1)', process_time
DATETIME DEFAULT CURRENT_TIMESTAMP);-- 5. ⑤CREATE TABLE
system_optimization_config ( id BIGINT PRIMARY KEY AUTO_INCREMENT
COMMENT 'ID', optimization_target VARCHAR(50) NOT NULL COMMENT '优化目标
/策略', brain_weight FLOAT NOT NULL COMMENT '脑权重(0-1)',
limb_weight FLOAT NOT NULL COMMENT '肢体权重(0-1)', cardiac_weight FLOAT
NOT NULL COMMENT '心脏权重(0-1)', neural_adjust_coeff FLOAT NOT NULL
COMMENT '神经调整系数', joint_response_threshold FLOAT NOT NULL COMMENT '关节
响应阈值', is_effective TINYINT DEFAULT 1 COMMENT '是否有效0-否/1-是');-- ⑤ INSERT
INTO system_optimization_config (optimization_target, brain_weight,
limb_weight, cardiac_weight, neural_adjust_coeff,
joint_response_threshold)VALUES ('', 0.4, 0.4, 0.2, 0.85, 0.1); ⑤
Python/C++/Java/MATLAB1. Python ⑤pythonimport tensorflow
as tfimport pymysqlimport jsonimport numpy as npfrom datetime import
datetime# ⑤class DBConnector: @staticmethod def get_connection():
return pymysql.connect( host="localhost", user="root", password="123456",
database="brain_limb_system", charset="utf8mb4",
cursorclass=pymysql.cursors.DictCursor )# ①②class
BrainNeuralModule: def __init__(self): self.model = self.build_neural_network()
self.conn = DBConnector.get_connection() def build_neural_network(self): # 4 ⑤
model = tf.keras.Sequential([ tf.keras.layers.Dense(512,
activation='relu', input_shape=(256,)), tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(256, activation='relu'), tf.keras.layers.Dense(128,
activation='sigmoid') ]) model.compile(optimizer='adam', loss='mse',
metrics=['accuracy']) return model def process_signal(self, perception_data): # ⑤
with self.conn.cursor() as cursor: cursor.execute("SELECT
physical_conductivity, bio_chem_params FROM brain_neural_system WHERE
neural_region=''" ) res = cursor.fetchone() conductivity =

```

```

res['physical_conductivity'] bio_params = json.loads(res['bio_chem_params']) # 
processed = self.model.predict(perception_data.reshape(1, -1))
[0] enzyme_activity = bio_params.get('enzyme_activity', 0.9) feedback_signal =
processed * conductivity * enzyme_activity return feedback_signal# 
class LimbJointModule: def __init__(self): self.conn =
DBConnector.get_connection() def control_joint(self, feedback_signal,
joint_name=""): # with self.conn.cursor() as cursor:
cursor.execute("SELECT motion_precision, torque_range FROM limb_joint_system
WHERE body_part=%s", (joint_name,)) res = cursor.fetchone() precision =
res['motion_precision'] torque = res['torque_range'] # joint_angle =
np.clip((feedback_signal.mean() * 120) / torque, 0, 120) # 0-120 control_cmd =
f"{joint_name},{joint_angle:.3f}°, {precision:.4f}mm" # with
self.conn.cursor() as cursor: cursor.execute(""" INSERT INTO multimodal_fusion
(application_scene, perception_types, control_command) VALUES (%s, %s, %s)
""", ("", "", control_cmd)) self.conn.commit() return
control_cmd# class CardiacModule: def __init__(self): self.conn =
DBConnector.get_connection() def monitor_supply(self): # with
self.conn.cursor() as cursor: cursor.execute("SELECT heart_rate, energy_supply
FROM cardiac_system ORDER BY collect_time DESC LIMIT 1") res =
cursor.fetchone() # if res['heart_rate'] > 120: new_supply =
max(0.5, res['energy_supply'] - 0.1) with self.conn.cursor() as cursor:
cursor.execute("UPDATE cardiac_system SET energy_supply=%s WHERE
id=(SELECT MAX(id) FROM cardiac_system)", (new_supply,)) self.conn.commit()
return f"{res['heart_rate']}/{new_supply:.1f}" return f"{
res['heart_rate']}/{res['energy_supply']:.1f}"# if
__name__ == "__main__": brain_module = BrainNeuralModule() limb_module =
LimbJointModule() cardiac_module = CardiacModule() # 256 
64+64+64+64 perception_data = np.random.randn(256) # 
feedback = brain_module.process_signal(perception_data) # joint_cmd =
limb_module.control_joint(feedback) # cardiac_status =
cardiac_module.monitor_supply() print("", joint_cmd, " | ", cardiac_status)
2. C++ #include <iostream>#include
<mysql_driver.h>#include <mysql_connection.h>#include
<json/json.h>#include <Eigen/Dense>#include <chrono>#include
<serial/serial.h> // using namespace std;using namespace
Eigen;using namespace chrono;// const string DB_HOST =
"localhost";const string DB_USER = "root";const string DB_PWD =
"123456";const string DB_NAME = "brain_limb_system";// ①②class
BrainNeuralModule {private: sql::mysql::MySQL_Driver* driver; sql::Connection*
conn; MatrixXd w1, w2, w3, w4; // 4 public: BrainNeuralModule() { // 
driver = sql::mysql::get_mysql_driver_instance(); conn = driver-
>connect(DB_HOST, DB_USER, DB_PWD); conn->setSchema(DB_NAME); // 
w1 = MatrixXd::Random(512, 256) * 0.05; w2 =
MatrixXd::Random(256, 512) * 0.05; w3 = MatrixXd::Random(128, 256) * 0.05;
w4 = MatrixXd::Random(64, 128) * 0.05; } VectorXd processSignal(const
VectorXd& data) { // sql::PreparedStatement* pstmt = conn-
>prepareStatement( "SELECT physical_conductivity, bio_chem_params FROM

```



```

brain_neural_system WHERE neural_region='"" ); sql::ResultSet* res = pstmt-
>executeQuery(); res->next(); float conductivity = res-
>getDouble("physical_conductivity"); string bioStr = res-
>getString("bio_chem_params"); //  Json::Reader reader; Json::Value
bioJson; reader.parse(bioStr, bioJson); float protein_activity = bioJson[""]
].asFloat(); //  VectorXd l1 = (w1 * data).unaryExpr([](double x)
{ return max(0.0, x); }); VectorXd l2 = (w2 * l1).unaryExpr([](double x) { return
max(0.0, x); }); VectorXd l3 = (w3 * l2).unaryExpr([](double x) { return max(0.0,
x); }); VectorXd l4 = (w4 * l3).unaryExpr([](double x) { return 1.0 / (1.0 + exp(-
x)); }); //  VectorXd feedback = l4 * conductivity * protein_activity;
delete pstmt; delete res; return feedback; } ~BrainNeuralModule() { delete conn;
}; //  ④class LimbHardwareModule {private: sql::Connection*
conn; serial::Serial* ser; //  public: LimbHardwareModule() { //
sql::mysql::MySQL_Driver* driver = sql::mysql::get_mysql_driver_instance(); conn
= driver->connect(DB_HOST, DB_USER, DB_PWD); conn->setSchema(DB_NAME);
//  ser = new serial::Serial("/dev/ttyUSB0", 9600,
serial::Timeout::simpleTimeout(100)); } void controlJoint(const VectorXd&
feedback, const string& joint) { //  sql::PreparedStatement* pstmt = conn-
>prepareStatement( "SELECT motion_precision, torque_range FROM
limb_joint_system WHERE body_part=?" ); pstmt->setString(1, joint);
sql::ResultSet* res = pstmt->executeQuery(); res->next(); float precision = res-
>getDouble("motion_precision"); float torque = res->getDouble("torque_range");
//  float angle = clamp(feedback.mean() * 180 / torque, 0.0f, 180.0f);
string cmd = "JOINT:" + joint + ",ANGLE:" + to_string(angle) + ",PRECISION:" +
to_string(precision); //  ser->write(cmd + "\n"); cout << "
" << cmd << endl; //  sql::PreparedStatement* logStmt = conn-
>prepareStatement( "INSERT INTO multimodal_fusion (application_scene,
perception_types, control_command) VALUES (?, ?, ?)" ); logStmt->setString(1, ""
); logStmt->setString(2, ""+ ""); logStmt->setString(3, cmd); logStmt-
>execute(); delete pstmt; delete res; delete logStmt; } ~LimbHardwareModule()
{ delete conn; delete ser; }; //  int main() { BrainNeuralModule
brain; LimbHardwareModule limb; //  256  VectorXd sensorData
= VectorXd::Random(256); //  auto start =
high_resolution_clock::now(); VectorXd feedback =
brain.processSignal(sensorData); auto end = high_resolution_clock::now();
double delay = duration_cast<microseconds>(end - start).count() / 1000.0; cout
<< " " << delay << "ms" << endl; limb.controlJoint(feedback, "");
return 0;}
3. Java  +  javaimport
com.mysql.cj.jdbc.MySQLDataSource;import
org.deeplearning4j.nn.conf.MultiLayerConfiguration;import
org.deeplearning4j.nn.conf.NeuralNetConfiguration;import
org.deeplearning4j.nn.conf.layers.DenseLayer;import
org.deeplearning4j.nn.conf.layers.OutputLayer;import
org.deeplearning4j.nn.multilayer.MultiLayerNetwork;import
org.nd4j.linalg.activations.Activation;import
org.nd4j.linalg.api.ndarray.INDArray;import org.nd4j.linalg.factory.Nd4j;import
org.nd4j.linalg.lossfunctions.LossFunctions;import org.json.JSONObject;import

```

```

java.sql.Connection;import java.sql.PreparedStatement;import
java.sql.ResultSet;import java.sql.SQLException;import
java.util.concurrent.ExecutorService;import java.util.concurrent.Executors;// ①②
class DBUtil { private static final String URL =
"jdbc:mysql://localhost:3306/brain_limb_system?
useSSL=false&serverTimezone=UTC"; private static final String USER = "root";
private static final String PWD = "123456"; public static Connection getConn()
throws SQLException { MysqlDataSource ds = new MysqlDataSource();
ds.setURL(URL); ds.setUser(USER); ds.setPassword(PWD); return
ds.getConnection(); } }// ③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
class BrainNeuralService { private
MultiLayerNetwork neuralNet; private Connection conn; public
BrainNeuralService() throws SQLException { // ①②③④⑤ MultiLayerConfiguration
config = new
NeuralNetConfiguration.Builder() .learningRate(0.001) .list() .layer(new
DenseLayer.Builder().nIn(256).nOut(512).activation(Activation.RELU).build()) .lay
er(new
DenseLayer.Builder().nIn(512).nOut(256).activation(Activation.RELU).build()) .lay
er(new
OutputLayer.Builder(LossFunctions.LossFunction.MSE) .nIn(256).nOut(64).activati
on(Activation.SIGMOID).build()) .build(); neuralNet = new
MultiLayerNetwork(config); neuralNet.init(); conn = DBUtil.getConn(); } public
INDArray processSignal(INDArray data) throws SQLException { // ⑥⑦⑧⑨⑩
PreparedStatement pstmt = conn.prepareStatement( "SELECT
physical_conductivity, bio_chem_params FROM brain_neural_system WHERE
neural_region=''" ); ResultSet rs = pstmt.executeQuery(); rs.next(); float
conductivity = rs.getFloat("physical_conductivity"); JSONObject bioJson = new
JSONObject(rs.getString("bio_chem_params")); float gene_activity =
bioJson.getFloat("FOXP2 "); // ⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
INDArray processed =
neuralNet.output(data); INDArray feedback =
processed.mul(conductivity).mul(gene_activity); rs.close(); pstmt.close(); return
feedback; } }// ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
class MultiSceneService { private
BrainNeuralService brainService; private Connection conn; private
ExecutorService executor = Executors.newFixedThreadPool(5); // ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
public MultiSceneService() throws SQLException { brainService = new
BrainNeuralService(); conn = DBUtil.getConn(); } // ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
private class
SceneTask implements Runnable { private String scene; private String joint;
public SceneTask(String scene, String joint) { this.scene = scene; this.joint =
joint; } @Override public void run() { try { // ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
INDArray data =
Nd4j.randn(1, 256); INDArray feedback = brainService.processSignal(data); // ①②
③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
PreparedStatement pstmt = conn.prepareStatement( "SELECT
motion_precision, torque_range FROM limb_joint_system WHERE body_part=?" );
pstmt.setString(1, joint); ResultSet rs = pstmt.executeQuery(); rs.next(); float
precision = rs.getFloat("motion_precision"); float torque =
rs.getFloat("torque_range"); // ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
float angle = (float) Math.min(150,
Math.max(0, feedback.meanNumber().doubleValue() * 150 / torque)); String cmd
= String.format("□□=%s,□□=%s,□□=%.2f°,□□=%.4fmm", scene, joint, angle,
precision); // ①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚㉛㉜㉝㉞㉟
PreparedStatement logStmt =

```

```

conn.prepareStatement( "INSERT INTO multimodal_fusion (application_scene,
perception_types, control_command) VALUES (?, ?, ?)" ); logStmt.setString(1,
scene); logStmt.setString(2, "000000"); logStmt.setString(3, cmd);
logStmt.execute(); System.out.println("00000000" + cmd); rs.close();
pstmt.close(); logStmt.close(); } catch (SQLException e)
{ e.printStackTrace(); } } } // 00000000 public void startMultiScene() { String[]
scenes = {"0000", "0000", "0000", "0000", "0000"}; String[] joints = {"00", "000",
"00", "000", "000"}; for (int i = 0; i < scenes.length; i++) { executor.submit(new
SceneTask(scenes[i], joints[i])); } executor.shutdown(); } } // 0000 public class
BrainLimbSynergySystem { public static void main(String[] args) throws
SQLException { MultiSceneService service = new MultiSceneService();
System.out.println("0000000000..."); service.startMultiScene(); } } 4. MATLAB 0000
000+000000matlab%% 000000 dbHost = 'localhost';dbUser = 'root';dbPwd =
'123456';dbName = 'brain_limb_system';% 000000 conn = database(dbName,
dbUser, dbPwd, 'com.mysql.cj.jdbc.Driver', ... ['jdbc:mysql://', dbHost, ':3306/',
dbName, '?useSSL=false&serverTimezone=UTC']);if ~isopen(conn) error('000000
');end%% 1. 0000000000①②- 0000 function feedback =
brain_neural_process(data, conn) % 000000 sqlQuery = "SELECT
physical_conductivity, bio_chem_params FROM brain_neural_system WHERE
neural_region='0000'"; dataNeural = fetch(conn, sqlQuery); conductivity =
dataNeural.physical_conductivity(1); biojson =
jsondecode(dataNeural.bio_chem_params{1}); enzyme = biojson.000000; % 00 4
000000MATLAB 0000000000 layers = [ fullyConnectedLayer(512, 'Activation', 'relu')
dropoutLayer(0.2) fullyConnectedLayer(256, 'Activation', 'relu')
fullyConnectedLayer(64, 'Activation', 'sigmoid') ]; options =
trainingOptions('adam', 'MaxEpochs', 1, 'Verbose', false); net =
trainNetwork(data', layers, options); % 0000+0000 processed = predict(net,
data'); feedback = processed * conductivity * enzyme;end%% 2. 0000000000④- 0
000 function control_cmd = limb_joint_control(feedback, joint, conn) % 000000
sqlQuery = sprintf("SELECT motion_precision, torque_range FROM
limb_joint_system WHERE body_part='%s'", joint); dataJoint = fetch(conn,
sqlQuery); precision = dataJoint.motion_precision(1); torque =
dataJoint.torque_range(1); % 000000 angle = min(180, max(0, mean(feedback) *
180 / torque)); control_cmd = sprintf('00=%s,00=%.3f°,00=%.4fmm', joint,
angle, precision); % 0000 sqlInsert = sprintf("INSERT INTO multimodal_fusion
(application_scene, perception_types, control_command) VALUES ('0000', '00+00',
'%s')", control_cmd); exec(conn, sqlInsert);end%% 3. 00000000⑤⑥% 00 256 0000
0000 perceptionData = randn(1, 256);% 000000 feedbackSignal =
brain_neural_process(perceptionData, conn);% 0000 controlCmd =
limb_joint_control(feedbackSignal, '00', conn);% 000000
figure;subplot(2,1,1);plot(perceptionData, 'b-', 'LineWidth', 1);title('000000
');xlabel('00'); ylabel('00');subplot(2,1,2);plot(feedbackSignal, 'r-', 'LineWidth',
1);title('000000');xlabel('00'); ylabel('0000');fprintf('00000000%s\n', controlCmd);%
% 00000000 close(conn); 000000000000 0000 000000 0000 00000000 00 256 0/00 64 00
4 000000000000≤3ms00000000≥100 0 CPU≥16 00GPU≥24GB0000 CUDA000000≥32GB
Python(TensorFlow)0C++(Eigen)0Java(DL4J)0MATLAB(DeepLearnToolbox) 000000
0000 0000≤0.001mm0000 0.05-10N·m000000≤1ms000000≥200 0 0000≤0.01°000000

```

采样率 $\geq 2000\text{Hz}$ 波特率 $\geq 9600\text{bps}$ 支持Eigen 矩阵 支持40-180 度/秒角速度
抖动 $\leq 0.1\%$ 分辨率 0.05 频率分辨率 $\geq 100\text{Hz}$ 相位分辨率 $\leq 0.5\%$ 支持Java Executors
支持4+ 线程 吞吐量 $\geq 1\text{GB/s}$ 延迟 $\leq 5\text{ms}$ 带宽 $\geq 2\text{Gbps}$ latency $\leq 2\text{ms}$
JSON 支持BLOB 数据类型 数据类型 0-1 精度 0.01 覆盖率 $\geq 30\%$ 支持 ≥ 3 线程
延迟 $\leq 1\text{ms}$ MySQL ≥ 8.0 支持多线程 支持 $\geq 2000\text{QPS}$ 支持 $\geq 1\text{TB}$ 数据
延迟 $\leq 0.5\text{ms}$ 支持IO $\geq 1\text{GB/s}$ 支持 $\geq 64\text{GB}$ MySQL 8.0+ JDBC/PyMySQL 支持 支持Protobuf 支持+ MATLAB 支持.mat 支持

●●●